Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Primožič

Kompresija slik s pomočjo linearne regresije in delitve slike na ploske sektorje

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matej Kristan

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Stiskanje slik je ključnega pomena v številnih aplikacijah in multimedijskih sistemih. Kompresijski standardi kot so JPEG in PNG so se dodobra uveljavili v vsakdanji rabi. Vendar pa so ti standardi izdelani za optimalno kompresijo splošnih slik, zato temeljijo na v naprej predpisanih baznih funkcijah. Te bazne funkcije so pod-optimalne za slike z lokalno ploskimi intenzitetnimi profili. V diplomski nalogi zato obdelajte problem stiskanja slik z lokalno ploskim intenzitetnim profilom. Pozornost posvetite matematični formulaciji in učinkovitosti algoritma. Razvit postopek kompresije primerjajte z enim od uveljavljenih tehnik stiskanja slik.

Avtor se zahvaljuje družini za vero in podporo med tekom študija, Mateju Gašperinu in Matjažu Payritsu za pomoč pri interpretaciji omejitev, kolegu Žigi Sajovicu za pomoč pri izpeljavi teorije, ter Mateju Kristanu za zaupanje in aktivno vodenje izdelave dela.

Kazalo

Povzetek

Naslov: Kompresija slik s pomočjo linearne regresije in delitve slike na ploske sektorje

Avtor: Gašper Primožič

V diplomskem delu predlagamo novo metodo izgubnega ali neizgubnega stiskanja slik. Slike so v njihovi osnovni obliki, v računalniku predstavljene kot dvodimenzionalna polja intenzitet treh različnih barvnih kanalov. Barvni model slike spremenimo v takšnega, ki minimizira kovarianco med intenzitetami vseh barvnih kanalov, hkrati pa varianco maksimizira na enem samem kanalu. S pomočjo sektorizacije, posamezne barvne kanale razrežemo na sektorje, katerega intenzitete so sebi podobne. Intenzitete vseh stolpcev in vrstic dobljenih sektorjev aproksimiramo s pomočjo regresijskih funkcij prve stopnje. Tako za vsak sektor slike, namesto vseh intenzitet, ohranimo le koeficiente, ki opisujejo prileżne premice in na ta način dosežemo kompresijo. Ker so lokacije intenzitet vrstice ali stolpca vnaprej znane, lahko postopek iskanja koeficientov tovrstnih premic okrajšamo. Prav tako pa metoda predstavlja tudi možnost paralelizacije korakov kompresije in dekompresije, saj se glavnina računanja izvaja z matrikami, za kar pa so moderni procesorji in grafične kartice specializirani. Predstavimo tudi evalvacijo predstavljene metode. S pomočjo numeričnih orodij ocenjevanja kakovosti rekonstrukcij in statistične analize pokažemo, da se prednosti metode izkažejo pri močno stisnjenih slikah, katere so po večini ploske.

Ključne besede: kompresija, računalnik, linearna regresija.

Abstract

Title: Image compression with linear regression and image segmentation to flat sectors

Author: Gašper Primožič

We propose a new approach for lossless and lossy image compression. Images are stored in computer in its primary form, as two dimensional fields of intensities of three color channels. We change the color model of an image to one which minimizes the covariance between the intensities of all channels, and maximizes the variance on a single one. With the help of sectorization, we split each of the color channels to sectors, whose intensities are self similar. We approximate the intensities of all sectors, by applying first degree regression functions, to all rows and columns of a sector. Compression is achieved by storing only the coefficients of the regressed functions and discarding the original intensities. The procedure of finding the regression function coefficients can be speeded up since the positions of the intensities in a sector are known beforehand. This method is also expandable for paralellisation of crucial steps in the algorithm, because the majority of the time consuming calculations are matrix-based. Modern day processors and graphics cards are made for these kinds of applications, which can drastically drop the compression and decompression times. Our proposed compression method is thoroughly analyzed. Numerical approaches and statistical analysis are utilized to show that the advantage of this method is with highly compressed flat images.

Keywords: compression, computer, linear regression.

Poglavje 1

Uvod

Zametki shranjevanja slike segajo globoko v človekovo zgodovino. Že pralju-dje so skicirali po stenah z različnimi barvami in motivi. Zatem se je pričela doba umetnosti in portretov, kjer so z namensko barvo in platnom risali ter upodabljali podobe ali ljudi.

Z razvojem optike se je začela pojavljati drzna ideja o zajemanju okolice na medij. Sprva so se razvile metode zajemanja fotografij s pomočjo hlapov srebrovega halida, katerega postopek je bil nevaren in dolgotrajen, hkrati pa drag. Želja po kakovostnih fotografijah pa je ostajala močna, a za tisti čas futuristična. Z razvojem prvih osebnih fotografskih naprav (Kodak camera [17]) pa so lahko fotografirali mnogi. Fotografije so zajemali na rolo celulo-znega traku, prevlečenem s fotosenzitivno kemikalijo, ki so ga nato poslali do razvijalca slik, kjer so s traku fotografijo presneli na papir in jo nato poslali uporabniku. A tovrstne fotografije so bile še zmeraj črnobele. Z ra-zvojem fotosenzitivnih kemikalij, ki reagirajo pri različnih valovnih dolžinah svetlobe, pa je bilo moč zajeti tudi barvo trenutka. Metoda se je razvijala in izboljševala vse do prihoda digitalnih kamer. Digitalne kamere namesto ce-luloznega traku za zajem uporabljajo fotoaktivne polprevodniške elemente, ki delujejo na principu fotoelektričnega efekta, pokrite z raznimi barvnimi filtri za singulacijo intenzitet barv. Prevladujoči tehnologiji tovrstnih sen-zorjev sta CMOS in CCD [7]. Sprva so bile tovrstne naprave slabe, saj so

bile metode izdelave takšnih senzorjev drage. Z razvojem fotolitografije se je razvijala tudi sodobna kamera, fotografije pa so zajemale vse več informacije, saj se je z izboljševanjem metod hkrati povečevalo število pikslov na senzorju. Ker je fotografija postajala virtualno vse večja, pomnilnik pa je bil omejen, so se začele razvijati tudi metode kompresiranja slik.

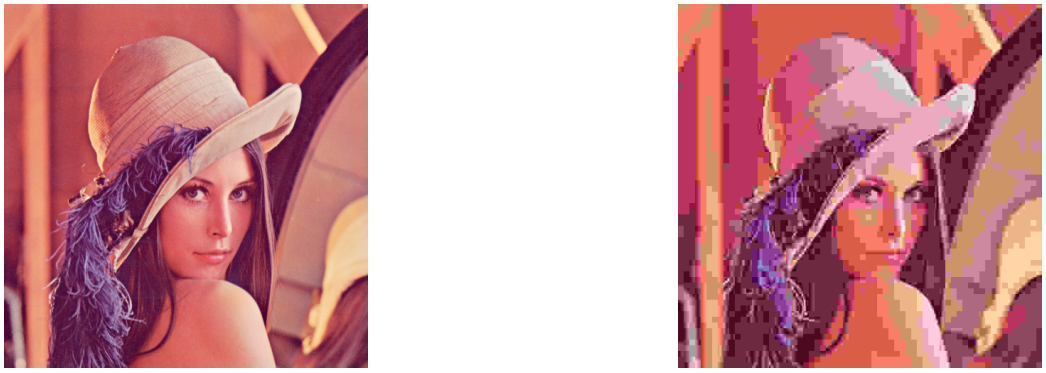
1.1 Motivacija

Slika v nekompresiranem formatu zaseda preveč prostora. Res je, da ohranja vse informacije o intenzitetah vseh pikslov, a naše oko je bolj občutljivo na velike spremembe intenzitet kot na majhne. Če hranimo vsako vrednost posebej, pa je tudi razpršenost intenzitet tako visoka, da niti brezizgubno entropsko kodiranje po intenzitetah ne prinaša večjih prednosti pri kompresiranju. Sliko je potrebno transformirati v signale, ki so si podobni in opišejo sliko dovolj dobro, da razlike niso očitne, hkrati pa jih je mogoče tudi dobro stisniti. Problem kompresiranja slik oziroma samih signalov je že dolgo prisoten, pravzaprav se je pojavil s pojavom medijev, rešitve pa so si v osnovi dokaj podobne.

1.2 Sorodna dela

Kakor slike, tudi ostale medije, kot so video in zvok, kompresiramo s pomočjo podobnih metod kompresiranja signalov. Kompresiramo jih v veliki večini [4] s pomočjo fourierjeve transformacije - specifično hitre fourierjeve transformacije FFT in diskretne kosinusne transformacije DCT. Obe metodi signal predstavita kot obteženo kombinacijo večih baznih sinusnih in kosinusnih signalov različnih frekvenc, ki enolično opišejo signale. Kot je bilo navedeno že v začetku, je oko bolj občutljivo na večje spremembe intenzitet, kot na manjše. Metode kompresiranja medijev se kompresije lotijo tako, da enostavno ignorirajo bazne signale, ki ne dodajo veliko informacije. To je razvidno takrat, ko poizkusimo sliko, video ali zvok stisniti preveč, kot to prikazuje

Slika 1.1. Pri zvoku se sliši, da ni na voljo celotnega razpona frekvenc, pri sliki in videu pa se to prepozna po različnih napakah in artefaktih v sliki, še najbolj v predelih kontrastnih sprememb. Tudi med nevronskimi mrežami so vse bolj popularni variacijski avtoenkoderji, ki stojijo na teoriji globokega učenja [9, 8] in Bayesianskih prijemov [10], slike pa s pomočjo pravilne vzpostavitve kodirne nevronske mreže stisnejo v množico intenzitet, ki jih lahko nato s pomočjo dekodirne nevronske mreže rekonstruiramo v približek slike, ki je takšne intenzitete povzročila.



(a) Original (b) Rekonstrukcija

Slika 1.1: Primer preveč stisnjene slike [3].

1.3 Prispevki naloge

Metoda kompresiranja, predstavljena v tem delu, se poslužuje lastnosti slik, da so sektorji slik velikokrat ploski. Ko fotografiramo npr. morsko obalo, smučišče, itd., velikokrat ujamemo v objektiv tudi nebo, ki je skoraj enakih intenzitet, oziroma se intenzitete spreminjajo zvezno.

Tovrstne ploske sektorje slik je moč rekonstruirati z manj informacij, kot bi jih potrebovali za opis vseh intenzitet. Zato se za vrstice in stolpce takšnih sektorjev izračunajo le čimbolj prileţne premice s pomočjo linearne regresije, katerih koeficiente nato hranimo. V nadaljevanju bodo koraki natančneje opisani.

1.4 Struktura naloge

Predstavljena naloga je razdeljena na pet poglavij. V Poglavju 1 je naloga opredeljena v splošnem smislu, Poglavje 2 pa opisuje pristope in algoritme, ki jih uporabljamo v nalogi. V začetku poglavja je opisana diskretna predstavitev slik v računalniku in nato še na kratko o barvnih modelih, ki jih za stiskanje uporabljamo. Sledi opis linearne regresije in njenih pohitritev, ki v implementaciji pridejo do izraza. Opisana je še sektorizacija, ki jo potrebujemo za izluščenje detajlov slike.

V Poglavju 3 je opisan način, kako prej omenjene in kasneje natančneje opisane algoritme implementiramo in uporabimo v prid kompresije. Sledi ji komplementarni algoritem dekompresije, kjer s pomočjo koeficientov, pridobljenih v fazi kompresije, rekonstruiramo približek originalu slike. Sledi še numerična analiza učinkovitosti stiskanja slik.

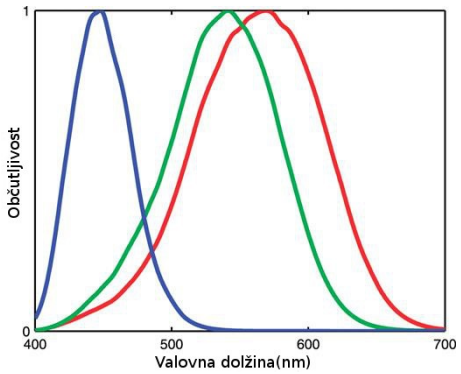
Eksperimentalna evalvacija in ocene kvalitete kompresije so opisane v Poglavju 4. V poglavju sta opisani dve metodi evalvacije. Numerična evalvacija kvalitete opisuje kvaliteto stiskanja slik, ki smo jo pridobili s pomočjo povprečne absolutne napake rekonstrukcije glede na original. Sledi ji še statistična analiza kvalitete, kjer je vzorec ljudi ocenjeval, kako dobro metoda ohranja informacijo v sliki pri različnih kompresijskih razmerjih. Poglavje 5 vsebuje zaključek in načrt za nadaljnje delo.

Poglavje 2

Osnovne metode

2.1 Barvni modeli

Slike so v računalniku v nekompresirani obliki shranjene kot polja intenzitet barvnih kanalov. Barvni kanali so po večini trije, torej rdeč, moder in zelen, saj so to tudi barve, na katere se trije tipi receptorjev za barvo v naših očeh najbolj odzivajo [2], kot je to predstavljeno na Sliki 2.1. Korelacija med tovrstnimi barvnimi kanali je visoka, kar pomeni, da se s spreminjanjem ene barve najverjetneje spreminjata tudi ostali dve. S tem vsak od kanalov nosi veliko takšne informacije, ki je vsem kanalom skupna, s tem pa se velikost slike veča. Naša želja je sliko čim bolj stisniti, zato je bolje, da barve vsakega piksla spremenimo v nek drug, bolj praktičen barvni model.



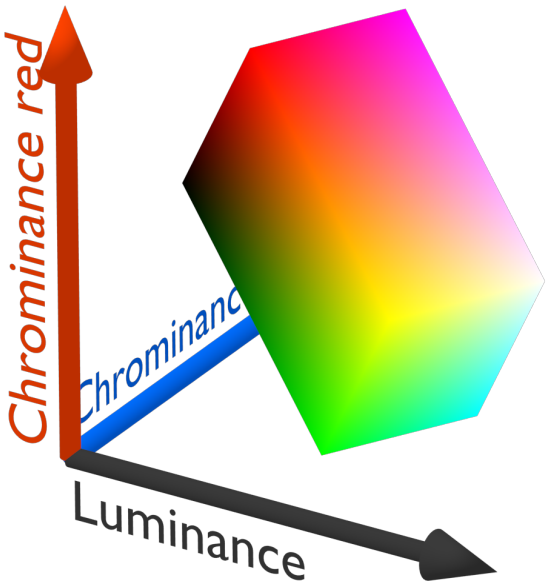
Slika 2.1: Odzivnost stožcev človeškega očesa glede na barvo svetlobe.

2.1.1 Barvni model RGB

Ekrani imajo v vsakem pikslu tri različne sektorje. To so rdeči, zeleni in modri sektor. Izbrani so bili na podlagi občutljivosti človeškega očesa, saj lahko na ta način dobro rekonstruiramo barvo, ki jo želimo na ekranu izrisati. Vsak piksel ima zato tri komponente, vsaka slika pa ima tri kanale, kjer vsak izmed kanalov predstavlja doprinos korespondenčne barve h končni barvi, ki jo izrišemo na ekran. Tak barvni model je v stroki poimenovan s kratico RGB.

2.1.2 Barvni model YCbCr

Barvni model YCbCr [11] je predstavitev neke barve s pomočjo komponent svetlosti in dveh komponent barvitosti. Gre za linearno transformacijo modela RGB, ki je nato normirana med 0 in 255. Ravno takšen barvni model nam pride najbolj prav, saj se izkaže, da je naše oko najbolj občutljivo na spremembe svetilnosti. Tako izoliramo najbolj entropsko bogat signal od tistih, ki ne hranijo veliko informacije o barvi in jih lahko zaradi tega tudi bolj stisnemo.



Slika 2.2: Vizualizacija YCbCr barvnega modela [12].

Izbran barvni model je normirana linearna transformacija, kjer za vsak piksel, zapisan v barvnem modelu RGB, izračunamo njegove YCbCr komponente, ki se izračunajo po enačbi

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.332 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{2.1}$$

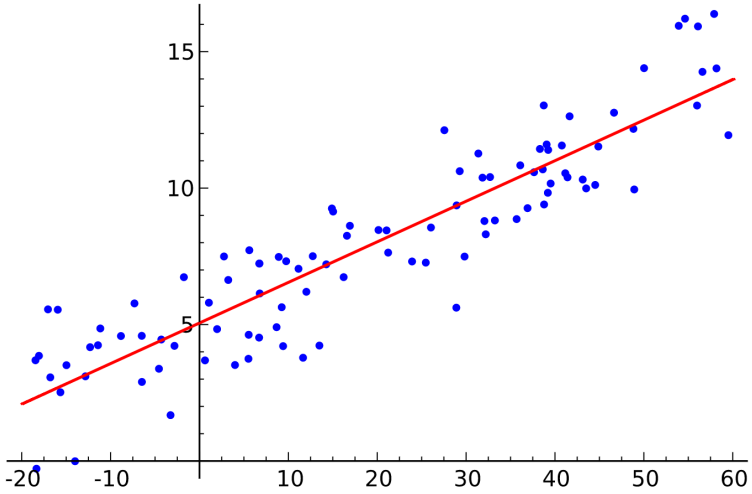
Postopek spremembe barvnega modela sliki je opisan v Algoritmu 1.

Algoritem 1 Spremeni barvni model vhodni sliki I in vrni rezultat O

```
1: function RGB2YCbCr(ImageRGB I)
2:     ImageYCbCr O
3:     for i = 1 → I.width do
4:         for j = 1 → I.height do
5:             PixelRGB p_i = I_{i,j}
6:             PixelYCbCr p_o
7:             p_o.Y  =0 +0.299·p_i.R +0.587·p_i.G +0.114·p_i.B
8:             p_o.Cb =128 −0.169·p_i.R −0.332·p_i.G +0.500·p_i.B
9:             p_o.Cr =128 +0.500·p_i.R −0.419·p_i.G −0.081·p_i.B
10:            O_{i,j} = p_o
11:    return O
```

2.2 Linearna regresija

Linearna regresija (LR) je v osnovi metoda prileganja linearnih modelov odvisnosti med odvisno spremenljivko in eno ali več neodvisnimi spremenljivkami za statistične analize podatkov, kakor je to predstavljeno v Sliki 2.3. LR ni omejena s številom naključnih spremenljivk, ki jih v dani iteraciji poskušamo aproksimirati. Postopek, opisan v diplomski nalogi, s pomočjo modela linearne premice opazuje in analizira odvisnost med intenziteto nekega piksla in njegovo lokacijo v vrstici oziroma stolpcu. Ker je lokacija pikslov diskretna in deterministična, nam to ponuja pohitritve, ki jih bomo kasneje v poglavju natančneje opisali in definirali. LR hkrati ponuja enostavno rešitev za brezizgubno kompresijo, saj linearna premica natančno opiše vse točke, oziroma za nas intenzitete, ki so linearno odvisne od lokacije v vrstici oziroma stolpcu.



Slika 2.3: Prileganje premice linearne regresije na splošne točke v dvodimenzionalnem koordinatnem sistemu [14].

2.2.1 Splošne enačbe LR za prileganje premic

V dvodimenzionalnem koordinatnem sistemu je poljubna točka opisana s pomočjo dveh koeficientov x in y. Enačba za premico v takšnem koordinatnem sistemu je definirana po enačbi

$$y = \alpha + \beta x, \tag{2.2}$$

kjer je x lokacija, y pa intenziteta ki jo želimo aproksimirati. Naša naloga je poiskati α in β, ki se čim bolj prilegata podatkom o intenzitetah y in lokaciji x, zato velja enačba za iskanje cenilke $\hat{\alpha}$

$$\hat{\alpha} = \bar{y} - \hat{\beta} \cdot \bar{x}, \tag{2.3}$$

kjer je

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \tag{2.4}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}. \tag{2.5}$$

Problema se v diplomski nalogi lotevamo s pomočjo metode najmanjših kvadratov. Ker ocenjujemo odvisno spremenljivko le glede na eno neodvisno

spremenljivko, si za iskanje naklona modela linearne funkcije lahko pomagamo s pomočjo variance in kovariance [15]. Kovarianca je mera, ki opisuje skupno spremenljivost dveh naključnih spremenljivk, varianca pa opisuje, kako razpršena je neka naključna spremenljivka. Zato veljata enačbi

$$C_{xy} = \sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y}), \tag{2.6}$$

$$C_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2, \tag{2.7}$$

zato lahko z njima izrazimo cenilko $\hat{\beta}$ kot

$$\hat{\beta} = \frac{C_{xy}}{C_{xx}}, \tag{2.8}$$

s tem pa tudi omogočimo izračun cenilke $\hat{\alpha}$, ki je izražena v enačbi (2.3).

2.2.2 Pohitritve

Splošne enačbe LR so primerne za točke, katerih x in y zajemata poljubne vrednosti iz množice realnih števil. V diplomski nalogi pa želimo aproksimirati vrstice in stolpce neke slike, zato lahko trdimo, da $\forall x_i \in \mathbb{N}$ in ko ocenjujemo zaporedje n komponent, zajema zalogo vrednosti $x_i = \{1, 2, ..., n\}$, kjer je $x_1 = 1, x_2 = 2, ...x_n = n$. Zato lahko zapišemo

$$\bar{x} = \frac{(n+1)}{2}, \tag{2.9}$$

za pomoč v nadaljevanju pa izpeljanke bernoullijeve enačbe za vsote n-tih potenc

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}, \tag{2.10}$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}. \tag{2.11}$$

Razvijemo C_{xx}:

$$C_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2, \tag{2.12}$$

$$= \sum_{i=1}^{n}(x_i^2 - 2x_i\bar{x} + \bar{x}^2), \tag{2.13}$$

$$= \sum_{i=1}^{n}(x_i^2) - \sum_{i=1}^{n}(2x_i\bar{x}) + \sum_{i=0}^{n-1}(\bar{x}^2), \tag{2.14}$$

$$= \sum_{i=1}^{n}(x_i^2) - 2\bar{x}\sum_{i=0}^{n-1}(x_i) + n\bar{x}^2, \tag{2.15}$$

$$= \sum_{i=1}^{n}(x_i^2) - 2\bar{x}\frac{n(n+1)}{2} + n\frac{(n+1)^2}{4}, \tag{2.16}$$

$$= \sum_{i=1}^{n}(x_i^2) - 2\frac{n+1}{2}\frac{n(n+1)}{2} + \frac{n(n+1)^2}{4}, \tag{2.17}$$

$$= \sum_{i=1}^{n}(x_i^2) - \frac{n(n+1)^2}{2} + \frac{n(n+1)^2}{4}, \tag{2.18}$$

$$= \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)^2}{4}, \tag{2.19}$$

$$= \frac{2n(n+1)(2n+1) - 3n(n+1)^2}{12}, \tag{2.20}$$

$$= \frac{4n^3 - 6n^2 + 2n - 3n^3 + 6n^2 - 3n}{12}, \tag{2.21}$$

$$= \frac{n^3 - n}{12}. \tag{2.22}$$

Kot je razvidno iz razvitja enačbe (2.12), se vrsta $C_{xx} = \sum_{i=0}^{n-1}(x_i - \bar{x})^2$ spremeni v enostavnejšo enačbo $C_{xx} = \frac{n^3-n}{12}$. To je polinom tretje stopnje, odvisen samo od dolžine oziroma števila pikslov, ki jih želimo v danem trenutku aproksimirati z LR. To nam drastično zmanjša čas računanja, saj za ta korak namesto časovne zahtevnosti $O(n)$ potrebujemo le $O(1)$.

Razvijemo še C_{xy}:

$$C_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}), \tag{2.23}$$

$$= \sum_{i=1}^{n} x_i y_i - x_i\bar{y} - \bar{x}y_i + \bar{x}\bar{y}, \tag{2.24}$$

$$= \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i\bar{y} - \sum_{i=1}^{n} \bar{x}y_i + n\bar{x}\bar{y}, \tag{2.25}$$

$$= \sum_{i=1}^{n} x_i y_i - \bar{y}\sum_{i=1}^{n} x_i - \bar{x}\sum_{i=1}^{n} y_i + n\bar{x}\bar{y}, \tag{2.26}$$

$$= \sum_{i=1}^{n} x_i y_i - \bar{y}\sum_{i=1}^{n} x_i - \cancel{\bar{x}n\bar{y}} + \cancel{n\bar{x}\bar{y}}, \tag{2.27}$$

$$= \sum_{i=1}^{n} x_i y_i - \bar{y}\sum_{i=1}^{n} x_i, \tag{2.28}$$

$$= \sum_{i=1}^{n} x_i y_i - \bar{y}\frac{n(n+1)}{2}, \tag{2.29}$$

$$= \sum_{i=1}^{n} x_i y_i - \frac{\cancel{n}(n+1)}{2}\sum_{i=1}^{n}\frac{y_i}{\cancel{n}}, \tag{2.30}$$

$$= \sum_{i=1}^{n} x_i y_i - \frac{n+1}{2}\sum_{i=1}^{n} y_i, \tag{2.31}$$

$$= \sum_{i=1}^{n} x_i y_i - \frac{n+1}{2}y_i, \tag{2.32}$$

$$= \sum_{i=1}^{n}(x_i - \frac{n+1}{2})y_i. \tag{2.33}$$

Tako smo skrajšali tudi enačbo za iskanje vrednosti C_{xy}. Še vedno ostaja vrsta, ki jo je treba sešteti, a ta z logičnega vidika mora ostati, saj drugače ne bi upoštevali vseh vrednosti, ki jih želimo v danem trenutku aproksimirati z LR. Tudi ta del nam kalkulacijski čas drastično zmanjša, saj nam ni potrebno vedno znova računati povprečij $\bar{x}$ in $\bar{y}$. Če pa razmislimo še o cenilkah $\hat{\alpha}$ in $\hat{\beta}$, lahko s pomočjo zgornjih enačb okrajšamo tudi te enačbe.

Razširimo enačbo za iskanje cenilke $\hat{\beta}$

$$\hat{\beta} = \frac{C_{xy}}{C_{xx}}, \tag{2.34}$$

$$= \frac{\sum_{i=1}^{n}(x_i - \frac{n+1}{2})y_i}{\frac{n^3-n}{12}}, \tag{2.35}$$

$$= \sum_{i=1}^{n} \frac{12(x_i - \frac{n+1}{2})}{n^3 - n} y_i, \tag{2.36}$$

$$= \sum_{i=1}^{n} (x_i \frac{12}{n^3 - n} - \frac{6}{n^2 - n}) y_i, \tag{2.37}$$

za cenilko $\hat{\alpha}$ pa

$$\hat{\alpha} = \bar{y} - \hat{\beta} \cdot \bar{x}, \tag{2.38}$$

$$= \sum_{i=1}^{n} \frac{y_i}{n} - \sum_{i=1}^{n} ((x_i \frac{12}{n^3 - n} - \frac{6}{n^2 - n}) y_i) \cdot \frac{n+1}{2}, \tag{2.39}$$

$$= \sum_{i=1}^{n} (\frac{n-1}{n^2 - n} + x_i \cdot \frac{-6}{n^2 - n} + \frac{3n+3}{n^2 - n}) y_i, \tag{2.40}$$

$$= \sum_{i=1}^{n} (x_i \cdot \frac{-6}{n^2 - n} + \frac{4n+2}{n^2 - n}) y_i. \tag{2.41}$$

Tako smo izračunali dve enačbi za izračun komponent $\hat{\alpha}$ in $\hat{\beta}$ modela premice, ki ga apliciramo na podatke, razporejene zaporedno, z razmakom velikosti 1. Če pozorno pogledamo enačbi, je razvidno, da se spreminjata le s spreminjanjem števila vhodnih podatkov, saj $x_i \in \{1, 2 \ldots i, \ldots n\}$. S tem so pohitritve LR za našo specifično uporabo zaključene.

2.3 Aproksimacije 1D polj s pomočjo LR

Kot smo videli v Poglavju 2.2.2, so se enačbe za izračun komponent premic drastično okrajšale. S tem je moč razviti tudi LR za enodimenzionalne objekte. Označimo tovrstna polja z vektorsko notacijo

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix}. \tag{2.42}$$

Naša želja je aproksimirati vektor $\mathbf{x}$ s pomočjo premice, ki bo s čim manjšo napako opisala komponente danega vektorja. Za pomoč predstavimo indeks elementov k kot vektor

$$\mathbf{k} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ k \\ \vdots \\ n \end{bmatrix}. \tag{2.43}$$

Ker smo v enačbi (2.34) okrajšali enačbe za iskanje komponente $\hat{\beta}$, v enačbi (2.38) pa za $\hat{\alpha}$, lahko to predstavimo kot linearno transformacijo. Za iskanje komponente premice $\hat{\beta}$ lahko izpostavimo

$$\boldsymbol{\gamma_n} = \mathbf{k} \cdot \frac{12}{n^3 - n} - \frac{6}{n^2 - n}, \quad \boldsymbol{\gamma_n} = \begin{bmatrix} 1 \cdot \frac{12}{n^3-n} - \frac{6}{n^2-n} \\ 2 \cdot \frac{12}{n^3-n} - \frac{6}{n^2-n} \\ \vdots \\ n \cdot \frac{12}{n^3-n} - \frac{6}{n^2-n} \end{bmatrix}, \tag{2.44}$$

zato lahko komponento $\hat{\beta}$ za aproksimacijo danega polja izrazimo kot skalarni produkt

$$\hat{\beta} = \boldsymbol{\gamma_n}^\top \cdot \mathbf{x}. \tag{2.45}$$

Za iskanje komponente premice $\hat{\alpha}$ pa lahko izpostavimo

$$\boldsymbol{\delta_n} = \mathbf{k} \cdot \frac{-6}{n^2 - n} + \frac{4n+2}{n^2-n}, \quad \boldsymbol{\delta_n} = \begin{bmatrix} 1 \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n} \\ 2 \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n} \\ \vdots \\ n \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n} \end{bmatrix}, \tag{2.46}$$

in izrazimo tudi $\hat{\alpha}$ kot skalarni produkt

$$\hat{\alpha} = \boldsymbol{\delta_n}^\top \cdot \mathbf{x}. \tag{2.47}$$

Iskanje komponent $\hat{\alpha}$ in $\hat{\beta}$ izrazimo kot linearno transformacijo

$$\begin{bmatrix} \boldsymbol{\delta_n}^\top \\ \boldsymbol{\gamma_n}^\top \end{bmatrix} \cdot \mathbf{x} = \begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \end{bmatrix}. \tag{2.48}$$

Izračuna vektorjev $\boldsymbol{\delta_n}$ in $\boldsymbol{\gamma_n}$ sta povzeta v Algoritmu 2. S tem smo poenostavili osnove za lažje razvijanje linearnih transformacij, potrebnih za kompresijo in dekompresijo sektorjev.

Algoritem 2 Izračun vektorjev $\boldsymbol{\delta_n}$ in $\boldsymbol{\gamma_n}$

function $get\delta$(Integer n)
    Vec $d(n)$
    for $i = 1 \to n$ **do**
        $d_i = i \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n}$
    return d
function $get\gamma$(Integer n)
    Vec $g(n)$
    for $i = 1 \to n$ **do**
        $g_i = i \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n}$
    return g

2.4 Aproksimacije 2D polj s pomočjo LR

Naša naloga je iz vsake vrstice in vsakega stolpca polja izračunati premico s koeficienti $\hat{\alpha}$ in $\hat{\beta}$, ki se bo s pomočjo metode najmanjših kvadratov najbolj prilegala vrednostim vrstice ali stolpca. Naj bo vhodno polje predstavljeno kot matrika $\mathbf{S}$, definirana kot

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \dots & s_{1n} \\ s_{21} & s_{22} & \dots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1} & s_{m2} & \dots & s_{mn} \end{bmatrix} \implies \mathbf{s} = \begin{bmatrix} s_{11} \\ \vdots \\ s_{1n} \\ s_{21} \\ \vdots \\ s_{mn} \end{bmatrix}. \tag{2.49}$$

Vektor $\mathbf{s}$ je transformacija matrike $\mathbf{S}$ v stolpični vektor, kamor smo po ključu od leve proti desni in od zgoraj navzdol, prepisali vse vrednosti matrike $\mathbf{S}$. Njegova višina je torej $m \times n$. Ker znamo iz LR za aproksimacije komponent 1D polja izračunati vektorje, s katerimi moramo množiti vhodne podatke, da dobimo $\hat{\alpha}$ in $\hat{\beta}$, lahko enačbe razširimo na tovrstno pretvorjene matrike. Ker je vrstic m in stolpcev n, iščemo pa $\hat{\alpha}$ in $\hat{\beta}$ za vsako vrstico in stolpec posebej, jih bomo označili z $\hat{\alpha}_i, \hat{\beta}_i; i \in \{1, 2, \dots, m+n\}$.

Če zapišemo iskane vrednosti kot vektor $\boldsymbol{\mu}$, t.j.,

$$\boldsymbol{\mu} = \begin{bmatrix} \hat{\alpha_1} \\ \dots \\ \hat{\alpha}_{m+n} \\ \hat{\beta}_1 \\ \dots \\ \hat{\beta}_{m+n} \end{bmatrix}, \tag{2.50}$$

lahko nato izrazimo enačbo za aproksimacijo vektorja $\boldsymbol{\mu}$ kot

$$\mathbf{M} \cdot \mathbf{s} = \boldsymbol{\mu}. \tag{2.51}$$

2.4.1 Zgradba matrike M

Matrika $\mathbf{M}$ iz enačbe (2.51) naj bo takšna matrika, da lahko z množenjem matrike $\mathbf{M}$ in transformacijo polja $\mathbf{s}$, izračunamo $\boldsymbol{\mu}$. Ker poznamo vektorske transformacije za iskanje $\hat{\alpha}$ in $\hat{\beta}$, poznamo tudi vse koeficiente, ki se bodo v matriki $\mathbf{M}$ pojavljali. Razsežnost $\mathbf{M}$ je $2(m+n) \times mn$.

Za začetek, si problem najprej razdelimo na podprobleme iskanja vrstičnih $\hat{\alpha}$, vrstičnih $\hat{\beta}$, stolpičnih $\hat{\alpha}$ in stolpičnih $\hat{\beta}$ komponent prileţnih premic.

Permutacijske in ciklične matrike

Permutacije v linearni algebri izražamo s permutacijskimi matrikami. Imenujmo takšne matrike z veliko črko $\mathbf{P}$. Pomembno je, da je $\mathbf{P}$ kvadratna binarna matrika, ki ima natanko en zapis števila 1 v vsaki vrstici in vsakem stolpcu, povsod drugod pa 0. S takšnimi matrikami lahko ustvarjamo poljubne permutacije vektorjev ali matrik. Velikost permutacijske matrike naj bo implicitno določena z višino vektorja, ki ga permutiramo. Primer

$$\mathbf{P} \cdot \mathbf{v} = \mathbf{u}, \tag{2.52}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} v1 \\ v2 \\ v3 \end{bmatrix} = \begin{bmatrix} v3 \\ v1 \\ v2 \end{bmatrix}. \tag{2.53}$$

Razvidno je, da je permutacijska matrika $\mathbf{P}$ matrika identitete[1], katere stolpci so bili krožno prestavljeni v levo za eno mesto, kar nam za eno mesto premakne tudi koeficiente vektorja, ki ga želimo permutirati. Tovrstne permutacijske matrike imenujemo krožne matrike [6]. Poimenujmo jih s $\mathbf{K}$, katere stolpci so definirani kot

$$\mathbf{K} = \begin{bmatrix} \mathbf{i}_2 & \mathbf{i}_3 & \dots & \mathbf{i}_n & \mathbf{i}_1 \end{bmatrix}, \tag{2.54}$$

kjer so vektorji $\mathbf{i}_j$ vektorji matrike identitete.

[1]Matrika identitete je kvadratna matrika z enicami po diagonali in ničlami drugod. Množenje vektorja ali matrike z matriko identitete vrne rezultat enak vhodnemu vekorju ali matriki.

Iskanje vrstičnih koeficientov $\hat{\alpha}$

Za iskanje $\hat{\alpha}$ potrebujemo vhodni vektor podatkov in $\boldsymbol{\delta_n}$, ki je odvisen od dolžine vrstice sektorja, ki ga želimo pretvoriti. Vrstični koeficient $\hat{\alpha_1}$ je odvisen samo od vrstice 1 v matriki $\mathbf{S}$, ki je transformirana v vektor $\mathbf{s}$ na mesta od 1 do n. Zato vektor $\mathbf{a}$ zapišemo kot kombinacijo komponent

$$a_k = \begin{cases} k \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n} & ; k \leq n \\ 0 & ; \text{sicer} \end{cases}, \tag{2.55}$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{m \cdot n} \end{bmatrix}. \tag{2.56}$$

Ker je generična vrstica i polja $\mathbf{S}$ zapisana v vektorju $\mathbf{s}$ z zamikom $(i-1)n$ mest, lahko iskanje poljubnega vrstičnega koeficienta $\hat{\alpha_i}$ izrazimo s krožno matriko

$$\mathbf{a}_i = \mathbf{K}^{(i-1)n} \cdot \mathbf{a}. \tag{2.57}$$

Ker je v matriki $\mathbf{S}$ število vrstic m, iščemo pa koeficient $\hat{\alpha_i}$ za vsako vrstico, lahko izrazimo matriko $\mathbf{A}$ kot kombinacijo krožno zamaknjenih vektorjev $\mathbf{a}$

$$\mathbf{A} = \begin{bmatrix} (\mathbf{K}^0 \cdot \mathbf{a})^\top \\ (\mathbf{K}^n \cdot \mathbf{a})^\top \\ \vdots \\ (\mathbf{K}^{(m-1)\cdot n} \cdot \mathbf{a})^\top \end{bmatrix}. \tag{2.58}$$

Iskanje stolpičnih koeficientov $\hat{\alpha}$

Stolpični koeficient $\hat{\alpha_1}$ je odvisen samo od stolpca 1 v matriki $\mathbf{S}$. Po definiciji je prvi stolpec matrike kombinacija prvih koeficientov vseh vrstic te matrike. Ker je vrstica i preslikana v vektor $\mathbf{s}$ z zamikom $(i-1)n$ mest, velja, da je stolpec 1 v vektorju $\mathbf{s}$ zapisan na mestih k, kjer k sovpada s pravilom

$$k \mod n \equiv 1; k \in \{1, 2, \ldots, m \cdot n\}, \tag{2.59}$$

zato lahko zapišemo vektor $\mathbf{b}$ kot kombinacijo komponent

$$b_k = \begin{cases} (1 + \frac{k-1}{n}) \cdot \frac{-6}{n^2-n} + \frac{4n+2}{n^2-n} & ; k \mod n \equiv 1 \\ 0 & ; \text{sicer} \end{cases}, \tag{2.60}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{m \cdot n} \end{bmatrix}. \tag{2.61}$$

Generični stolpični koeficient $\hat{\alpha_j}$ je odvisen samo od stolpca j v matriki $\mathbf{S}$, ki pa je po definiciji kombinacija j-tih koeficientov vseh vrstic. Zato si lahko pomagamo s krožno matriko

$$\mathbf{b}_j = \mathbf{K}^{(j-1)} \cdot \mathbf{b}. \tag{2.62}$$

Ker vemo, da je v matriki $\mathbf{S}$ število stolpcev n, iščemo pa koeficient $\hat{\alpha_j}$ za vsak stolpec, lahko izrazimo matriko B za iskanje stolpičnih koeficientov $\hat{\alpha_j}$ kot kombinacijo krožno zamaknjenih vektorjev $\mathbf{b}$

$$\mathbf{B} = \begin{bmatrix} (\mathbf{K}^0 \cdot \mathbf{b})^\top \\ (\mathbf{K}^1 \cdot \mathbf{b})^\top \\ \vdots \\ (\mathbf{K}^{(n-1)} \cdot \mathbf{b})^\top \end{bmatrix}. \tag{2.63}$$

Iskanje vrstičnih koeficientov $\hat{\beta}$

Podobno, kot iskanje vrstičnih koeficientov $\hat{\alpha}_i$, računamo tudi $\hat{\beta}_i$, le s to razliko, da uporabimo druge koeficiente in sicer tiste, ki smo jih definirali v Poglavju 2.3 aproksimacij enodimenzionalnih polj za računanje $\hat{\beta}$

$$c_k = \begin{cases} k \cdot \frac{12}{n^3-n} - \frac{6}{n^2-n} & ; k \leq n \\ 0 & ; \text{sicer} \end{cases}, \tag{2.64}$$

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{m \cdot n} \end{bmatrix}. \tag{2.65}$$

Zato lahko matriko za računanje vrstičnih koeficientov $\hat{\beta}$, zapišemo kot kombinacijo vektorjev $\mathbf{c}$ s krožnim zamikom, kot smo to storili v Poglavju 2.4.1 za računanje vrstičnih koeficientov $\hat{\alpha}$

$$\mathbf{C} = \begin{bmatrix} (\mathbf{K}^0 \cdot \mathbf{c})^\top \\ (\mathbf{K}^n \cdot \mathbf{c})^\top \\ \vdots \\ (\mathbf{K}^{(m-1)\cdot n} \cdot \mathbf{c})^\top \end{bmatrix}. \tag{2.66}$$

Iskanje stolpičnih koeficientov $\hat{\beta}$

Podobno, kot za iskanje stolpičnih koeficientov $\hat{\alpha}_j$, računamo tudi $\hat{\beta}_j$, le s to razliko, da uporabimo druge koeficiente in sicer tiste, ki smo jih definirali v Poglavju 2.3

$$d_k = \begin{cases} (1 + \frac{k-1}{n}) \frac{12}{m^3-m} - \frac{6}{m^2-m} & ; k \mod n \equiv 1 \\ 0 & ; \text{sicer} \end{cases}; \tag{2.67}$$

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{m \cdot n} \end{bmatrix}. \tag{2.68}$$

Zato lahko matriko za računanje stolpičnih koeficientov $\hat{\beta}$ zapišemo kot kombinacijo vektorjev $\mathbf{d}$ s krožnim zamikom, kot smo to storili v Poglavju 2.4.1 za računanje stolpičnih koeficientov $\hat{\alpha}$

$$\mathbf{D} = \begin{bmatrix} (\mathbf{K}^0 \cdot \mathbf{d})^\top \\ (\mathbf{K}^1 \cdot \mathbf{d})^\top \\ \vdots \\ (\mathbf{K}^{(n-1)} \cdot \mathbf{d})^\top \end{bmatrix}. \tag{2.69}$$

Ker poznamo zgradbo vektorja $\boldsymbol{\mu}$, lahko matriko $\mathbf{M}$ zapišemo kot kombinacijo štirih matrik

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \end{bmatrix}. \tag{2.70}$$

Tako smo ustvarili vsa potrebna orodja za transformacijo matrik v koeficiente prileţnih premic vrstic in stolpcev. Vredno je omeniti, da se matrika $\mathbf{M}$ spreminja zgolj s spreminjanjem dimenzij matrike $\mathbf{S}$, zato je pravzaprav sploh ni treba vedno znova računati, ampak si jo lahko za različne razsežnosti matrike $\mathbf{S}$ vnaprej izračunamo in zapomnimo. Izračun matrike je povzet v Algoritmu 3, izračun vektorja $\boldsymbol{\mu}$ pa v Algoritmu 4.

Algoritem 3 Izračunaj matriko M izračun vrstičnih koeficientov $\hat{\alpha}$ in $\hat{\beta}$ matrike razsežnosti $m \times n$

1: **function** GETROWMRX(Integer m, n, Vec $\mathbf{v}$)
2: Vec $\mathbf{a}(m \cdot n)$
3: $\mathbf{a}_{1 \to n}$=$\mathbf{v}$
4: Mrx $\mathbf{A}(m, m \cdot n)$
5: **for** $i = 1 \to m$ **do**
6: $\mathbf{A}_{i,.}$=$\mathbf{K}^{n \cdot (i-1)} \cdot \mathbf{a}$
7: **return A**
8: **function** GETCOLMRX(Integer m, n, Vec $\mathbf{v}$)
9: Vec $\mathbf{a}(m \cdot n)$
10: **for** $i = 1 \to m$ **do**
11: $\mathbf{a}_{(i-1) \cdot n+1}$=$\mathbf{v}_i$
12: Mrx $\mathbf{A}(n, m \cdot n)$
13: **for** $i = 1 \to n$ **do**
14: $\mathbf{A}_{i,.}$=$\mathbf{K}^{i-1} \cdot \mathbf{b}$
15: **return A**
16: **function** GETM(Integer m, n)
17: Mrx $\mathbf{M}(2 \cdot (m + n), m \cdot n)$
18: $\mathbf{M}_{1 \to n,.}$ =getRowMrx$(m, n, get\gamma(n))$
19: $\mathbf{M}_{(n+1) \to (m+n),.}$ =getColMrx$(m, n, get\gamma(m))$
20: $\mathbf{M}_{(m+n+1) \to (m+2 \cdot n),.}$ =getRowMrx$(m, n, get\delta(n))$
21: $\mathbf{M}_{(m+2 \cdot n+1) \to (2 \cdot (m+n)),.}$ =getColMrx$(m, n, get\delta(m))$
22: **return M**

Algoritem 4 Izračunaj vektor $\boldsymbol{\mu}$ za vhodno 2D polje *img*

```
1: function GETμ(Mrx img)
2:     Integer m = img.height
3:     Integer n = img.width
4:     Mrx M = getM(m, n)
5:     Vec s(2 · (m + n))
6:     for i = 1 → m do
7:         for j = 1 → n do
8:             s_((i−1)·n)+j = img_m,n
9:     return M · s
```

2.5 Sektorizacija slik

Slike so sestavljene iz različnih odtenkov in intenzitet, kar pa premice slabo opišejo. Zato moramo sliko sektorizirati na ploske sektorje, ki jih nato lažje opišemo s pomočjo premic.

2.5.1 Sektorizacija v eni dimenziji

Za lažje razumevanje potrebne metode za sektorizacijo slik v dveh dimenzijah, se najprej omejimo na enodimenzionalni problemski prostor. Prileżna premica je definirana po enačbi

$$f(x) = \alpha + \beta x. \tag{2.71}$$

Prileżno premico izračunamo s pomočjo LR, katere postopek smo opisali v Poglavju 2.3. Premica, ki jo izračunamo, ima v vsaki točki aproksimacijsko napako, katero lahko izrazimo kot povprečno kvadratno napako po enačbi

$$e = \sum_{i=1}^{n} \frac{(y_i - f(i))^2}{n}. \tag{2.72}$$

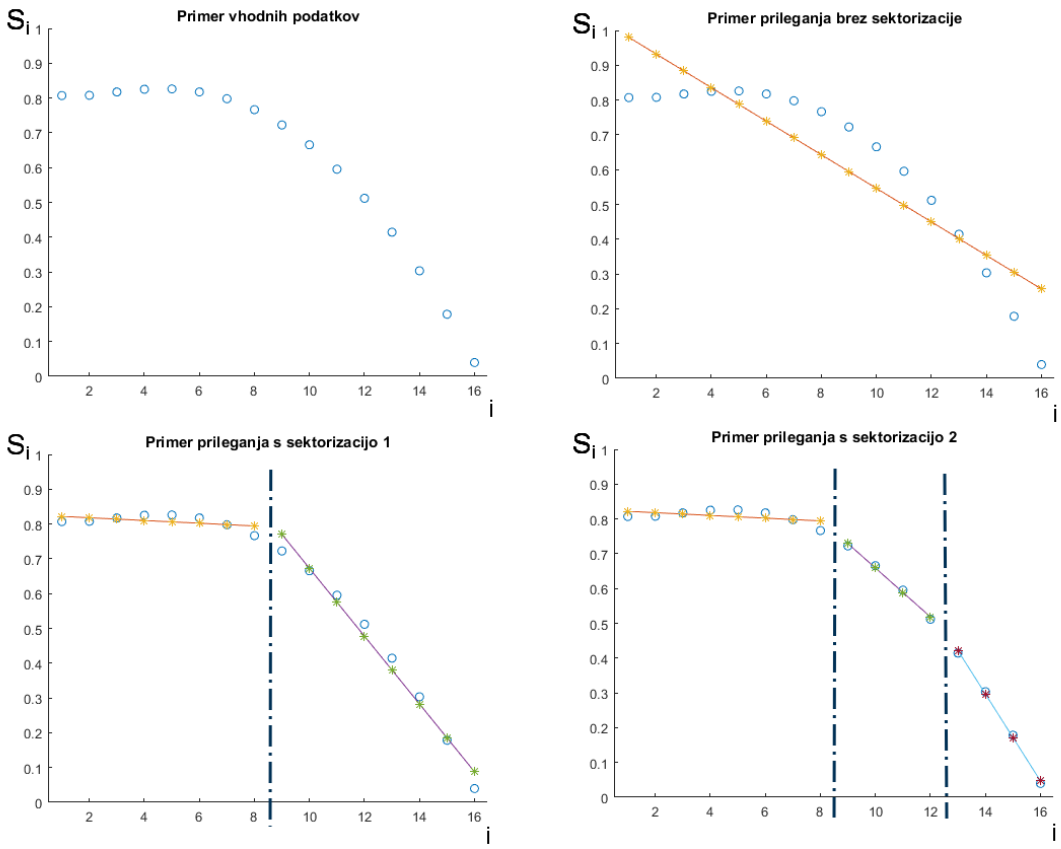
Ker želimo z našo metodo slike stiskati tudi izgubno, si definiramo dopustno napako e_{margin} in pogoj za nadaljno sektorizacijo izrazimo kot

$$e > e_{\text{margin}}. \tag{2.73}$$

Če premica opiše vhodne podatke z dovolj majhno napako, oziroma $e \leq e_{\text{margin}}$, si zapomnimo koeficiente premice. Če pa je napaka med premico in vhodnimi podatki prevelika, premica ne predstavlja dovolj dobre rekonstrukcije. Zato moramo vhodne podatke sektorizirati na manjše podsektorje. To storimo tako, da vhodne podatke razdelimo na dve podskupini (dva sektorja), na katerih nato izvajamo enak postopek.

Razdelimo jih na dva enaka dela. Vsak sektor smatramo kot neodvisno entiteto, zato se lahko sektorizacija na podsektorjih nadaljuje in posledično tudi ustavi na različnih nivojih. Postopek izvajamo vse dokler je napaka večja od mejne napake, oziroma dokler sektor zajema več kot dva vhodna podatka, saj gre premica, ki se najbolj prilega natanko dvema točkama, skozi njiju. Izračun napake je povzet v Algoritmu 5.

Grafi v Sliki 2.4 prikazujejo primere sektorizacij. Predstavljeni so štirje grafi, katerih abscisna os predstavlja odmik, ordinatna os pa intenziteto. Zgoraj levo so predstavljeni vhodni podatki, na katerih izvajamo postopek sektorizacije, kot je to predstavljeno v ostalih treh grafih. Premice prikazujejo rekonstrukcijsko funkcijo, križci na njej pa rekonstruirane intenzitete. Zgoraj desno je predstavljen primer, kjer začetno premico smatramo kot dovolj dobro rekonstrukcijo. Ostala dva primera s črtkano črto prikazujeta postopek sektorizacije, kjer se podatki razrežejo na dva podsektorja. Na sliki spodaj desno je predstavljen še primer, kjer se sektorizacija ustavi na različnih nivojih.



Slika 2.4: Primer sektorizacije in prileganja premic z različnimi dopustnimi rekonstrukcijskimi napakami e_{margin}.

Algoritem 5 Izračunaj napako med projekcijo in 1 dimenzionalnim poljem y

```
1: function getErr1D(Vec y, Decimal α, Decimal β)
2:     Integer n = y.length
3:     Decimal err = 0
4:     for i = 1 → n do
5:         err = err + (y_i − (α + β · i))
6:     err = err/n
7:     return err
```

2.5.2 Sektorizacija v dveh dimenzijah

Slike, ki jih želimo aproksimirati, so dvodimenzionalni signali. Brez sektorizacije bi naša rekonstrukcija zelo slabo aproksimirala intenzite, kot je to razvidno v Sliki 2.5, kjer je na levi strani predstavljen original, na desni pa prikaz rekonstrukcije brez uporabe metode sektorizacije.

Slika 2.5: Primer rekonstrukcije brez uporabe metode sektorizacije.

V Poglavju 2.5.1 smo opisali, kako sektorizirati enodimenzionalni signal s pomočjo delitve signala in dopustne rekonstrukcijske napake. Ker aproksimiramo dvodimenzionalna polja intenzitet, moramo ta postopek ekstrapolirati na dve dimenziji.

Implementacija 2D sektorizacije

Sliko smatramo kot skupek vrstic in stolpcev. Na vsaki vrstici in stolpcu izvedemo aproksimacijo premice, s pomočjo katere izračunamo napako po enačbi (2.72). Če je napaka katerekoli vrstice oz. stolpca večja od mejne napake, je to indikator, da se intenzitete spreminjajo prehitro, oziroma, da s pomočjo premice ne moremo dovolj dobro opisati intenzitet slike. Sliko moramo zato razrezati na štiri podkvadrante, katerih ogljišča se srečajo na sredini slike. Na vsakem od podsektorjev izvedemo enako operacijo, kot smo jo opisali za celotno sliko. Sektorje režemo tako dolgo, dokler niso vse napake manjše od mejne napake, oziroma dokler ne pridemo do sektorja velikosti 2×2. Takrat se s pomočjo metode LR za iskanje primerne premice sektor opiše brez izgub, saj izračunane premice vrstic in stolpcev natanko opišejo dejanske intenzitete. Pogoj za nadaljnjo sektorizacijo dvodimenzionalnih struktur lahko zato definiramo kot

$$\max(e_i) > e_{\text{margin}}. \tag{2.74}$$

Izračun napake za dvodimenzionalna polja je povzet v Algoritmu 6. Na Sliki 2.6 je prikazan primer sektorizacije slike. Slika je razrezana na sektorje in podsektorje, ki se nato smatrajo kot ploski. Razlike v intenzitetah vsakega sektorja niso velike, zato lahko sektorje opišemo s pomočjo postopka prileganja premic, kot smo to opisali za dvodimenzionalna polja v Poglavju 2.4.

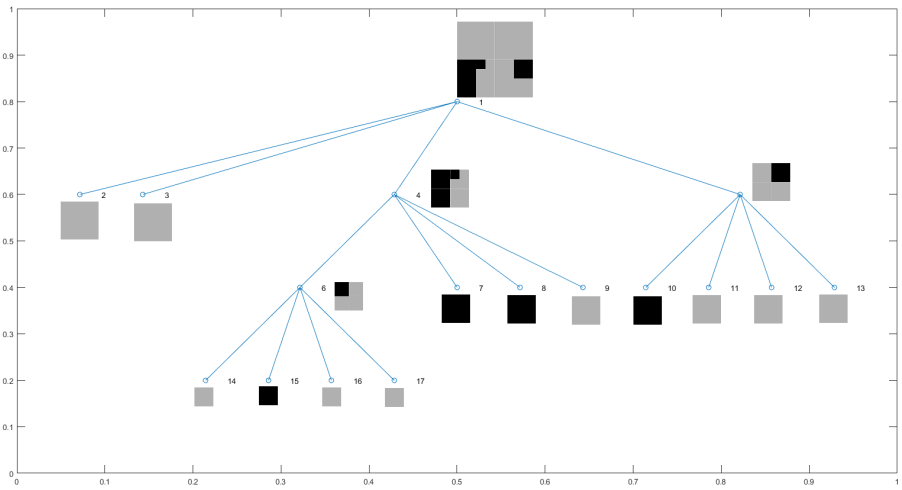
Slika 2.6: Primer rezanja slike na ploske sektorje.

Algoritem 6 Izračunaj napako med projekcijami $\boldsymbol{\mu}$ in 2 dimenzionalnim poljem *img*

```
1: function getErr2D(Mrx img, Vec μ)
2:     Integer m = img.height
3:     Integer n = img.width
4:     Decimal err_max = 0
5:     for i = 1 → m do
6:         err_new = getErr1D(img_{i,.}, μ_i, μ_{m+n+i})
7:         if err_new > err_max then
8:             err_max = err_new
9:     for i = (m + 1) → m + n do
10:        err_new = getErr1D(img_{.,i}, μ_i, μ_{m+n+i})
11:        if err_new > err_max then
12:            err_max = err_new
13:    return err_max
```

2.5.3 Predstavitev sektorizacije

Ker sliko razrežemo na štiri sektorje, za katere se nato po enakem postopku odločamo o nadaljni sektorizaciji, si lahko dobljeno sektorizacijsko zaporedje predstavljamo kot drevesno podatkovno strukturo z vejitveno stopnjo štiri. Slika 2.7 predstavlja grafični prikaz takšnega drevesa. Koren drevesa predstavlja celotna slika. Vsak od naslednikov vozlišča predstavlja enega od podsektorjev. Vozlišča so tisti sektorji, ki smo jih delili na podsektorje, listi pa tisti, ki smo jih uspeli uspešno opisati s prileţnimi premicami. Pomembno je omeniti, da si moramo za shranjevanje slike zapomniti le koeficiente prileţnih premic, ki opisujejo sektorje, ki so v listih drevesa.



Slika 2.7: Primer sektorizacijskega drevesa.

Poglavje 3

Kompresija slik z lokalnimi ploskvami

V tem Poglavju bomo natančneje opisali uporabo metod predstavljenih v Poglavju 2, hkrati pa bomo predstavili inverzno funkcijo, s katero lahko rekonstruiramo sliko, ki smo jo stisnili.

3.1 Kompresija

Zaenkrat smo opisali teoretični model vseh potrebnih postopkov za kompresijo slik, ki jih je potrebno združiti v smiselno celoto. Kompresija slik, predstavljena v diplomskem delu, smatra sliko kot skupek treh barvnih kanalov, ki jih smatramo kot osnovne sektorje. Za sektor izračunamo prileżne premice vrstic in stolpcev. Na vsakem sektorju izvajamo postopek sektorizacije. Pogoj za nadaljno sektorizacijo je izpolnjen, ko maksimalna napaka rekonstruiranih vrstic in stolpcev posameznega sektorja presega mejno napako.

3.1.1 Diskretizacija drevesne strukture

Ker poznamo drevesno strukturo postopka sektorizacije, lahko s pomočjo premega obhoda po drevesu in predstavitve elementov drevesa z binarnimi

vrednostmi, shranimo drevo v zelo majhnem prostoru. Naj bit 1 predstavlja vozlišče oz. sektorizacijo, bit 0 pa list drevesa oz. končni sektor. Tako dobimo binarni zapis drevesa, ki ga lahko s pomočjo premega obhoda in poznavanja stopnje vejitve ponovno zgradimo v fazi rekonstrukcije osnovne slike.

3.1.2 Diskretizacija koeficientov premic

Koeficienti pridobljenih premic so zapisani v 32-bitnem zapisu `float32`. Naša želja je čimbolj zmanjšati velikost transformiranega zapisa slike. Intenzitete ki jih opisujemo, so predstavljene v 8-bitnem zapisu `uint8` in zajemajo vrednosti med 0 in 255. Tudi rekonstruirana slika mora zajemati takšne vrednosti, zato si raje zapomnimo približne intenzitete na začetku $y_1 = \hat{\alpha} + \hat{\beta} \cdot 1$ in na koncu $y_n = \hat{\alpha} + \hat{\beta} \cdot n$ premice, ki opisuje podatke vrstice oz. stolpca sektorja. Tako lahko vsako prileţno premico dobro opišemo z dvemi 8-bitnimi števili y_1 in y_n. S pomočjo premega obhoda po drevesni strukturi pa lahko koeficiente premic sektorjev zapisujemo na disk z dobro definiranim zaporedjem, kar nam bo v fazi rekonstrukcije koristilo kot pravilo za branje koeficientov.

3.2 Učinkovitost

Mera učinkovitosti je kvantitativna mera uspešnosti stiskanja. Predstavlja razmerje med stisnjenimi in nestisnjenimi podatki. V splošnem

$$R = \frac{\text{SIZE}_{\text{uncompressed}}}{\text{SIZE}_{\text{compressed}}};\ R \in (0, \infty), \tag{3.1}$$

kjer višja vrednost predstavlja boljšo učinkovitost stiskanja. Kompresijski algoritem je kombinacija rezanja slike na ploske sektorje in aproksimacije sektorjev s pomočjo LR. Kompresijski korak pride v poštev šele takrat, ko namesto $a \cdot b$ intenzitet shranimo le koeficiente $\boldsymbol{\mu}$ za določen sektor. Ker je velikost $\boldsymbol{\mu}$ odvisna od višine in širine sektorja, ki ga v danem trenutku obravnavamo, učinkovitost kompresiranja naraste z rastjo sektorja. Velikost $\boldsymbol{\mu}$ za sektor velikosti $a \cdot b$ je enaka

$$|\boldsymbol{\mu}| = 2 \cdot (a + b). \tag{3.2}$$

Ker $|\boldsymbol{\mu}|$ predstavlja število koeficientov, ki aproksimirajo dani sektor, lahko izrazimo učinkovitost stiskanja posameznega sektorja kot

$$R_{\text{sector}} = \frac{a \cdot b}{2 \cdot (a + b)}. \tag{3.3}$$

Razberemo lahko, da bo učinkovitost stiskanja danega sektorja R_{sector} večja od 1, kadar

$$2 \cdot (a + b) < a \cdot b. \tag{3.4}$$

Sliko smo v fazi kompresije razdelili po barvnih kanalih in vsakega od njih razrezali na ploske sektorje, katerih velikosti so znane. Ker moramo kompresirati zgolj tiste sektorje, ki jih ne režemo dalje, lahko to izrazimo s pomočjo sektorizacijskega drevesa. Shraniti moramo koeficiente sektorjev $\boldsymbol{\mu}$, ki so listi v drevesu. Prešteti moramo torej število listov na vsakem nivoju drevesa. Ker je velikost korenskega sektorja oziroma celotnega barvnega kanala enaka $m \times n$, njegovega naslednika pa $\frac{m}{2} \times \frac{n}{2}$, itd., lahko izrazimo število koeficientov vseh sektorjev

$$\mathrm{N}_{\text{coefficients}} = \sum_{i=1}^{d} (p_i \cdot 2 \cdot (\frac{m}{2^{i-1}} + \frac{n}{2^{i-1}})), \tag{3.5}$$

kjer je d maksimalna globina sektorizacijskega drevesa, p_i pa število listov na i-tem nivoju. Za rekonstrukcijo potrebujemo še sektorizacijsko drevo. Drevo smo v Poglavju 3.1.1 predstavili s pomočjo premega obhoda in zapisovanja ničel in enic. Vsako vozlišče v drevesu k velikosti drevesa prispeva natanko en bit. Če preštejemo število vozlišč v danem drevesu, je rezultat velikost drevesa

$$\mathrm{SIZE}_{\text{tree}} = \sum_{i=1}^{d} (p_i). \tag{3.6}$$

Velikost stisnjenega barvnega kanala je seštevek velikosti drevesa in števila koeficientov pomnoženih s številom bitov, ki jih vsak koeficient zajema. Iz-

razimo jo lahko kot

$$\text{SIZE}_{\text{colorchannel}} = \text{SIZE}_{\text{tree}} + 8 \cdot \text{N}_{\text{coefficients}}, \tag{3.7}$$

$$= \sum_{i=1}^{d}(p_i) + 8 \cdot \sum_{i=0}^{d}(p_i \cdot 2 \cdot (\frac{m}{2^{i-1}} + \frac{n}{2^{i-1}})), \tag{3.8}$$

$$= \sum_{i=1}^{d}(p_i + 8 \cdot p_i \cdot 2 \cdot (\frac{m}{2^{i-1}} + \frac{n}{2^{i-1}})), \tag{3.9}$$

$$= \sum_{i=1}^{d} p_i \cdot (1 + 16 \cdot (\frac{m}{2^{i-1}} + \frac{n}{2^{i-1}})). \tag{3.10}$$

Slika je s pomočjo barvnega modela YCbCr predstavljena v treh ločenih kanalih. Vsakega od njih smo stisnili po enakem postopku. Torej je skupna velikost stisnjene slike enaka seštevku velikosti vseh stisnjenih barvnih kanalov

$$\text{SIZE}_{\text{image}} = \text{SIZE}_{\text{Y}} + \text{SIZE}_{\text{Cb}} + \text{SIZE}_{\text{Cr}}. \tag{3.11}$$

Ker podatke o sliki še entropsko kodiramo s pomočjo algoritma LZW [16], lahko rezultat stisnjenega objekta izrazimo kot

$$\text{SIZE}_{\text{entropic}} = \text{LZW}(\text{SIZE}_{\text{image}}). \tag{3.12}$$

S tem lahko izrazimo skupno učinkovitost stiskanja

$$R = \frac{\text{SIZE}_{\text{entropic}}}{3 \cdot 8 \cdot m \cdot n}. \tag{3.13}$$

Celoten postopek kompresije slike je povzet v Algoritmu 7.

Algoritem 7 Izračunaj kompresijske koeficiente slike rgb, s pomočjo mejne napake e_{margin}

```
1: Decimal e_margin = const
2: function recursiveCompress(Mrx x)
3:     Integer m = x.height
4:     Integer n = x.width
5:     BoolVec t(1)
6:     Vec μ = getμ(x)
7:     if getErr2D(x,μ) ≤ e_margin then
8:         t_1 = True
9:     else
10:        BoolVec a,b,c,d
11:        Vec μ_a,μ_b,μ_c,μ_d
12:        a,μ_a = recursiveCompress(x_{1→m/2,1→n/2})
13:        b,μ_b = recursiveCompress(x_{m/2+1→m,1→n/2})
14:        c,μ_c = recursiveCompress(x_{1→m/2,n/2→n})
15:        d,μ_d = recursiveCompress(x_{m/2+1→m,n/2→n})
16:        t_1 = False
17:        t = t.append(a).append(b).append(c).append(d)
18:        μ = μ_a.append(μ_b).append(μ_c).append(μ_d)
19:    return t, μ
20: function compress(ImageRGB rgb)
21:    ImageYCbCr img = RGB2YCbCr(rgb)
22:    BoolVec t_Y,t_Cb,t_Cr
23:    Vec c_Y,c_Cb,c_Cr
24:    t_Y, c_Y = recursiveCompress( img_Y)
25:    t_Cb, c_Cb = recursiveCompress( img_Cb)
26:    t_Cr, c_Cr = recursiveCompress( img_Cr)
27:    return t_Y.append(t_Cb).append(t_Cr), c_Y.append(c_Cb).append(c_Cr)
```

3.3 Dekompresija

Sliko smo v Poglavju 3.1 pretvorili v barvni model, ki ohranja večino informacije na enem kanalu, nato smo s pomočjo sektorizacije rezrezali vse kanale na ploske sektorje in s pomočjo LR za aproksimacije dvodimenzionalnih polj pretvorili še vsak sektor v komponente $\hat{\alpha}_i$ in $\hat{\beta}_i$ ki opisujejo vrstice in stolpce sektorjev. Vse te podatke o transformacijah smo si shranili in jih entropsko kodirali. Postopek dekompresije mora zato vsak korak kompresije opraviti v obratnem vrstnem redu in v njegovi inverzni obliki.

3.3.1 Inverz sektorizacije

Sektorizacijsko drevo, potrebno za sektorizacijo barvnega kanala, smo si v postopku kompresije shranili s pomočjo premega obhoda in shranjevanja binarnih vrednosti, ki opisujejo strukturo drevesa. Ker poznamo velikost originala vsake slike, lahko isti postopek premega obhoda ponovimo, da zgradimo strukturo sektorizacijskega drevesa. Ko iz binarnega zapisa preberemo bit 0, to pomeni, da se sektor, na katerem smo, ne sektorizira. Ker smo v koraku kompresiranja koeficiente premic shranili v istem vrstnem redu, lahko tudi te preberemo po vrsti, kakor smo jih shranili. Če preberemo bit 1, pa to pomeni, da smo v postopku kompresije sektor razrezali. Razrežemo ga na enak način, kot smo to storili v koraku sektorizacije. Na ta način si zgradimo drevo sektorjev, hkrati pa za vsak sektor poznamo koeficiente, ki ga opisujejo.

3.3.2 Inverzna LR za 2D polja

Ker poznamo osnovno enačbo, po kateri smo izračunali komponente $\hat{\alpha}$ in $\hat{\beta}$ za vrstice in stolpce danega sektorja (2.51) in pri dekompresiji poznamo $\boldsymbol{\mu}$,

iščemo pa $\mathbf{s}$, ga lahko izrazimo kot

$$\begin{aligned} \mathbf{M}^{-1}\mathbf{M}\mathbf{s} &= \mathbf{M}^{-1}\boldsymbol{\mu}, \\ \mathbf{I}\mathbf{s} &= \mathbf{M}^{-1}\boldsymbol{\mu}, \\ \mathbf{s} &= \mathbf{M}^{-1}\boldsymbol{\mu}. \end{aligned} \tag{3.14}$$

Inverz matrike M

Matrika $\mathbf{M}$, definirana pri kompresiranju sektorjev v Poglavju 2.4.1, je nekvadratna matrika, zato inverza nima. A s pomočjo razcepa matrike po lastnih vrednostih [5], za kar se v stroki uporablja kratica SVD, lahko izračunamo matriko $\mathbf{M}^{\dagger}$, ki predstavlja psevdoinverz matrike $\mathbf{M}$, pridobljen po metodi najmanjših kvadratov in bo služila kot nadomestilo inverzu $\mathbf{M}^{-1}$. Psevdoinverz se izračuna z enačbo

$$\mathbf{M}^{\dagger} = \mathbf{V} \cdot \mathbf{\Sigma}^{-1} \cdot \mathbf{U}^{\top}, \tag{3.15}$$

zato velja

$$\mathbf{s} \approx \mathbf{M}^{\dagger}\mathbf{b}. \tag{3.16}$$

S tem aproksimiramo koeficiente vektorja, za katerega razsežnosti osnovne matrike oziroma sektorja poznamo. Zato ga transformiramo v matriko, ki bo predstavljala rekonstrukcijo sektorja, po ključu vsakih n koeficientov se prične nova vrstica

$$\mathbf{s} = \begin{bmatrix} s_{11} \\ s_{12} \\ \vdots \\ s_{1n} \\ s_{21} \\ \vdots \\ s_{mn} \end{bmatrix} \implies S = \begin{bmatrix} s_{11} & s_{12} & \dots & s_{1n} \\ s_{21} & s_{22} & \dots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1} & s_{m2} & \dots & s_{mn} \end{bmatrix}. \tag{3.17}$$

Transformacija vektorja je opisana v Algoritmu 8.

Algoritem 8 Izračunaj 2D polje *img* za vhodni vektor $\boldsymbol{\mu}$

```
1: function VEC2MRX(Vec v, Integer m,n)
2:     Mrx a(m, n)
3:     for i = 1 → m do
4:         for j = 1 → n do
5:             a_{i,j} = v_{(i−1)·n+m}
6:     return a
```

3.3.3 Pretvorba barvnega modela

Ker smo za barvni model izbrali transformacijsko matriko, s katerim množimo intenzitete RGB vsakega piksla, da dobimo YCbCr barvni model, lahko poiščemo njen inverz, s katerim moramo množiti intenzitete YCbCr barvnega modela

$$\begin{aligned} \mathbf{Q} \cdot \mathbf{X} &= \mathbf{Y}, \\ \mathbf{Q}^{-1}\mathbf{Q}\mathbf{X} &= \mathbf{Q}^{-1}\mathbf{Y}, \\ \mathbf{X} &= \mathbf{Q}^{-1}\mathbf{Y}. \end{aligned} \tag{3.18}$$

Matriko $\mathbf{Q}$ smo definirali v Poglavju 2.2. Ker je matrika $\mathbf{Q}$ kvadratna bazna matrika ortogonalnih vektorjev, vemo, da je njen inverz možen in ga lahko tudi zapišemo

$$\mathbf{Q}^{-1} = \begin{bmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.770 & 0.000 \end{bmatrix}, \tag{3.19}$$

ter ga apliciramo na vse intenzitete rekonstruirane slike. Pretvorba barvnega modela sliki je povzeta v Algoritmu 9.

Algoritem 9 Spremeni barvni model vhodni sliki I in vrni rezultat O

```
1: function YCbCr2RGB(ImageYCbCr I)
2:     ImageRGB O
3:     for i = 1 → I.width do
4:         for j = 1 → I.height do
5:             PixelYCbCr p_i = I_{i,j}
6:             PixelRGB p_o
7:             p_i.Cb − 128
8:             p_i.Cr − 128
9:             p_o.R =1.000·p_i.Y +0.000·p_i.Cb +1.402·p_i.Cr
10:            p_o.G =1.000·p_i.Y −0.344·p_i.Cb −0.714·p_i.Cr
11:            p_o.B =1.000·p_i.Y +1.722·p_i.Cb +0.000·p_i.Cr
12:            O_{i,j} = p_o
13:    return O
```

Opisali smo vsa potrebna orodja za rekonstruiranje slike iz objektov, ki smo jih shranili v fazi kompresije. Algoritem 10 je povzetek vseh orodij v smiselno celoto.

Algoritem 10 S pomočjo binarnega zapisa drevesnih struktur in koeficientov rekonstruiraj sliko

```
1: BoolVec t
2: Vec c
3: function recursiveDecompress(Integer m,n)
4:     Mrx a(m,n)
5:     if t.pop() == True then
6:         Vec μ = c.pop(2 · (m + n))
7:         Mrx M = getM(m,n)
8:         Mrx U,S,V = SVD(M)
9:         Mrx M^+ = V· S^{-1} · U
10:        a= Vec2Mrx(M^+ · μ)
11:    else
12:        a_{1→m/2,1→n/2} = recursiveDecompress(t, c, m/2, m/2)
13:        a_{m/2+1→m,1→n/2} = recursiveDecompress(t, c, m/2, m/2)
14:        a_{1→m/2,n/2→n} = recursiveDecompress(t, c, m/2, m/2)
15:        a_{m/2+1→m,n/2→n} = recursiveDecompress(t, c, m/2, m/2)
16:    return a
17: function decompress(BoolVec treeData, Vec coeffs, Integer m,n)
18:    t = treeData
19:    c = coeffs
20:    ImgYCBCR img(m,n)
21:    img.Y=recursiveDecompress(m,n)
22:    img.Cb=recursiveDecompress(m,n)
23:    img.Cr=recursiveDecompress(m,n)
24:    return YCbCr2RGB(img)
```

Poglavje 4

Evalvacija

Ker diplomsko delo opisuje algoritem za stiskanje slik, je najprimerneje, da se kakovost stiskanja kvantificira in ovrednoti. To poglavje povzema numerično in eksperimentalno evalvacijo naše metode.

4.1 Podatkovna zbirka

Šest različnih slik smo stisnili s pomočjo algoritma JPEG, s tremi različnimi kakovostnimi razredi. Kakovostne razrede smo izbrali s pomočjo razmerja R, opisanega v Poglavju 3.2. Za lažjo interpretacijo pomena si definirajmo Q

$$Q = \frac{1}{R}, \tag{4.1}$$

ki predstavlja inverz učinkovitostnemu razmerju, oziroma velikost stisnjenih objektov glede na nestisnjeno sliko. Visoko kakovostni razred predstavljajo slike, kjer je

$$Q_{\mathrm{HQ}} = 0.9 \pm 0.025, \tag{4.2}$$

oziroma, kjer je velikost stisnjenih objektov enaka $90\% \pm 2.5\%$ velikosti nestisnjene slike. Razred, ki predstavlja srednjo kakovost, so slike, kjer je

$$Q_{\mathrm{SQ}} = 0.5 \pm 0.025, \tag{4.3}$$

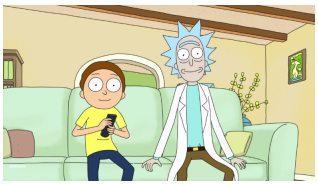
nizko kakovostni razred pa predstavljajo slike, kjer je

$$Q_{\mathrm{LQ}} = 0.1 \pm 0.025. \tag{4.4}$$

S pomočjo kompresije slik z lokalnimi ploskvami (LP) smo stisnili enakih šest slik tako, da so bile velikosti stisnjenih objektov enake velikosti stisnjenih slik s pomočjo JPEG z dovoljeno napako petih odstotkov. Rezultat je šestintrideset slik, ki predstavljajo osemnajst parov slik, torej šest originalov, stisnjenih z JPEG in LP pri treh različnih kakovostnih razmerjih. Slika 4.1 predstavlja originale slik, na katerih smo nato izvajali različne kompresijske algoritme pri različnih kakovostnih razmerjih.

(a) Slika 1

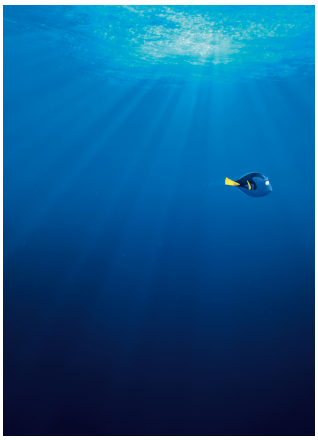
(b) Slika 2



(c) Slika 3

(d) Slika 4

(e) Slika 5



(f) Slika 6

Slika 4.1: Originali slik

Originale slik smo izbrali na podlagi raznolikosti motivov in stilov. Slika 4.2 predstavlja stisnjene slike visoke kakovosti. Prva vrstica predstavlja slike, stisnjene s pomočjo JPEG, druga pa s pomočjo metode LP. Slika 4.3 predstavlja stisnjene slike srednjega kakovostnega razreda in kot zadnje, Slika 4.4

predstavlja slike, stisnjene z visoko prostorsko prioriteto, kjer smo slike želeli stisniti v čim manj prostora.

4.2 Numerična evalvacija

Za vse slike, ki smo jih stisnili, smo numerično ocenili napako, ki nastane pri izgubnem stiskanju slik. Napako smo izračunali s pomočjo povprečne absolutne napake

$$\text{MAE} = \frac{1}{3 \cdot m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} |O_{P_{i,j}} - R_{R_{i,j}}| + |O_{G_{i,j}} - P_{G_{i,j}}| + |O_{B_{i,j}} - P_{B_{i,j}}|, \quad (4.5)$$

kjer O predstavlja original slike, P pa rekonstrukcijo le-te. Absolutne povprečne napake MAE intenzitet rekonstrukcij vseh slik, stisnjenih s pomočjo algoritma JPEG pri različnih kakovostnih razmerjih, so predstavljene v Tabeli 4.1, absolutne povprečne napake MAE intenzitet rekonstrukcij vseh slik, stisnjenih s pomočjo LP pri različnih kakovostnih razmerjih, pa v Tabeli 4.2.

Slika	Visoka kakovost(Q_{HQ})	Srednja kakovost(Q_{SQ})	Nizka kakovost(Q_{LQ})
Slika 1	0,482	4,082	16,288
Slika 2	9,166	9,744	13,544
Slika 3	0,394	3,989	17,784
Slika 4	0,321	2,050	12,504
Slika 5	1,182	4,543	8,521
Slika 6	0,102	1,526	12,053

Tabela 4.1: Povprečne absolutne napake intenzitet rekonstrukcij s pomočjo algoritma JPEG.

Slika	Visoka kakovost(Q_{HQ})	Srednja kakovost(Q_{SQ})	Nizka kakovost(Q_{LQ})
Slika 1	3,089	7,179	19,598
Slika 2	9,278	12,396	15,522
Slika 3	2,515	7,398	12,623
Slika 4	1,252	1,186	1,490
Slika 5	4,620	5,562	9,683
Slika 6	1,210	1,886	2,526

Tabela 4.2: Povprečne absolutne napake intenzitet rekonstrukcij s pomočjo LP.

Iz tabel lahko razberemo, da je pri visoki kakovosti napaka MAE algoritma JPEG manjša pri vseh slikah, oziroma, da rekonstrukcija s pomočjo LP rekonstruira sliko slabše kot algoritem JPEG. Pri srednji kakovosti kompresije, so razlike med JPEG in PS manjše in iz njih ni moč postaviti hipoteze o rekonstrukciji z manjšo napako, medtem ko izračuni MAE pri nizko kakovostnih slikah potrjujejo, da metoda LP z visoko prostorsko prioriteto, rekonstruira ploske slike veliko natančneje kot JPEG. To je najrazvidneje v Sliki 4, ki sta izpostavljeni v Sliki 4.5, kjer je $\mathrm{MAE}_{\mathrm{LQ}_{\mathrm{JPEG}}} = 12,504$, medtem, ko je $\mathrm{MAE}_{\mathrm{LQ}_{\mathrm{LP}}} = 1,490$.

Dobljene rezultate za algoritem JPEG smo povprečili po posameznih razredih kakovosti in s pomočjo LR izračunali trend povprečne absolutne napake, ki nastane z nižanjem kakovosti stisnjene slike Q

$$\mathrm{MAE}_{\mathrm{JPEG}}(Q) = 13.76 - 14.38Q. \tag{4.6}$$

Isti izračun smo izvedli tudi na rezultatih meritev kompresiranja s pomočjo LP

$$\mathrm{MAE}_{\mathrm{LP}}(Q) = 10.74 - 8.22Q. \tag{4.7}$$

Rezultati so predstavljeni v Sliki 4.6. Dobljene trende smo primerjali med seboj in razbrali, da absolutna napaka algoritma JPEG raste v odvisnosti s kakovostjo kompresiranja z večjim naklonom, kot trend absolutne napake

kompresije z metodo LP, kar je predstavljeno na Sliki 4.7. Interval in prelomno točko, kjer $\mathrm{MAE}_{\mathrm{JPEG}}$ postane večji od $\mathrm{MAE}_{\mathrm{LP}}$, smo izračunali z neenačbo

$$\begin{aligned} \mathrm{MAE}_{\mathrm{JPEG}}(Q) &\leq \mathrm{MAE}_{\mathrm{LP}}(Q), \\ 13.76 - 14.38Q &\leq 10.74 - 8.22Q, \\ Q &\leq 0.49. \end{aligned} \tag{4.8}$$

4.3 Eksperimentalna evalvacija

Vseh osemnajst parov slik smo predstavili množici sedemintridesetih ljudi. Za vsako kompresijsko razmerje vsake slike sta bili na voljo dve sliki, od katerih je boljšo moral izbrati uporabnik. Pari so bile enake slike, stisnjene z enakimi kakovostnimi razmerji, od tega ena s pomočjo algoritma JPEG, druga pa s pomočjo kompresije LP. Vse odločitve smo zabeležili na spletni strani in ustvarili statistiko.

Tabela 4.3 predstavlja deleže ljudi, ki so kot boljšo kategorizirali sliko, ki je bila stisnjena s pomočjo metode LP. Vredno je izpostaviti, da so se največje prednosti metode LP izkazale pri močno stisnjenih slikah, ki so po večini površine ploske, kjer pristop s pomočjo algoritma DCT ne prinaša dovolj dobrih rekonstrukcij, metdem ko kompresija s pomočjo ploskih sektorjev takšne slike dobro stisne in posledično tudi rekonstruira.
Največji delež se je izkazal pri Sliki 6, stisnjeni z nizko kakovostjo, katere primerjava je izpostavljena v Sliki 4.8. Algoritem JPEG prelive barv pri nizki kakovosti zanemarja in cilja na čim boljše približke barv, medtem ko kompresija z metodo LP tovrstne prelive dobro ohrani. Skoraj enaki rezultati so se izkazali tudi za Sliko 4, katere primerjava je predstavljena v Sliki 4.5. Ponovno gre za plosko sliko, katere intenzitete se dobro opiše s pomočjo premic.

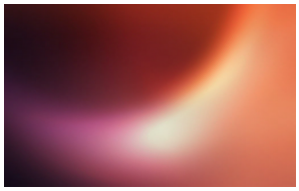
Slika	Visoka kakovost(Q_{HQ})	Srednja kakovost(Q_{SQ})	Nizka kakovost(Q_{LQ})
Slika 1	0,600	0,000	0,000
Slika 2	0,300	0,033	0,033
Slika 3	0,467	0,000	0,000
Slika 4	0,433	0,500	0,800
Slika 5	0,300	0,533	0,500
Slika 6	0,533	0,100	0,833

Tabela 4.3: Deleži vzorca, ki so kot boljšo rekonstrukcijo kategorizirali kompresijo s ploskimi sektorji.

(a) Slika 1

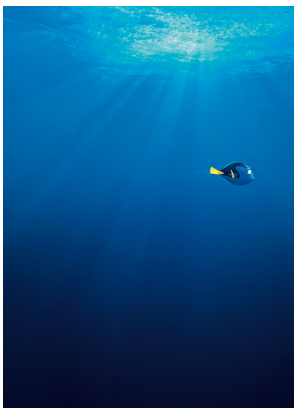
(b) Slika 2

(c) Slika 3



(d) Slika 4

(e) Slika 5 (LP)

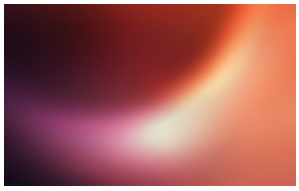


(f) Slika 6

(g) Slika 1

(h) Slika 2

(i) Slika 3

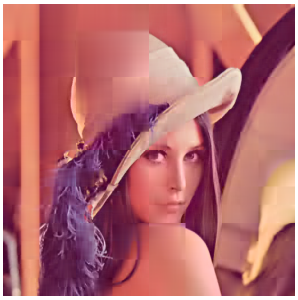


(j) Slika 4

(k) Slika 5 (JPEG)

(l) Slika 6

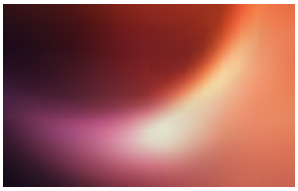
Slika 4.2: Visoka kakovost



(a) Slika 1

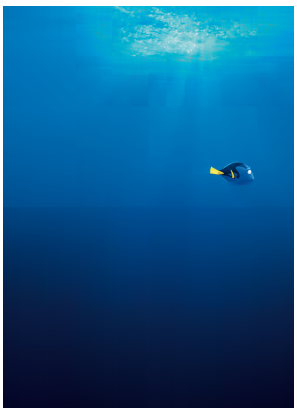
(b) Slika 2

(c) Slika 3



(d) Slika 4

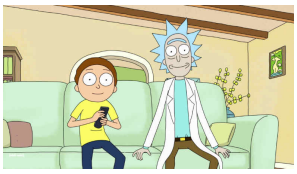
(e) Slika 5 (LP)



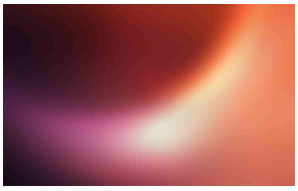
(f) Slika 6

(g) Slika 1

(h) Slika 2

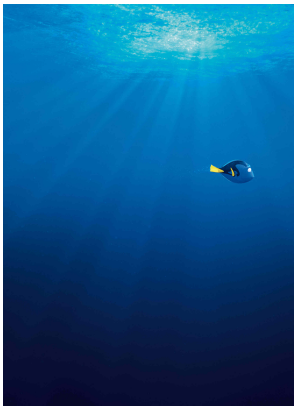


(i) Slika 3



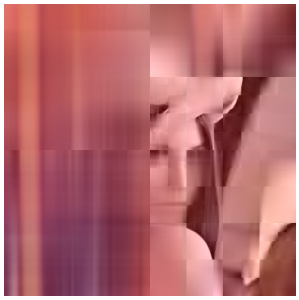
(j) Slika 4

(k) Slika 5 (JPEG)



(l) Slika 6

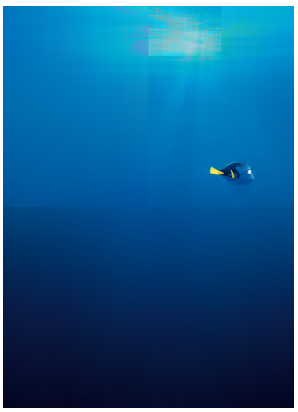
Slika 4.3: Srednja kakovost



(a) Slika 1

(b) Slika 2

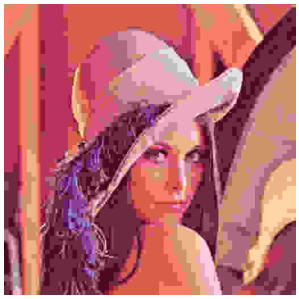
(c) Slika 3



(f) Slika 6

(d) Slika 4

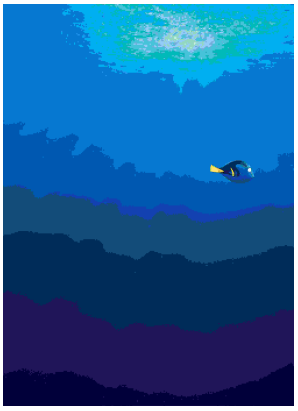
(e) Slika 5 (LP)



(g) Slika 1

(h) Slika 2

(i) Slika 3



(l) Slika 6

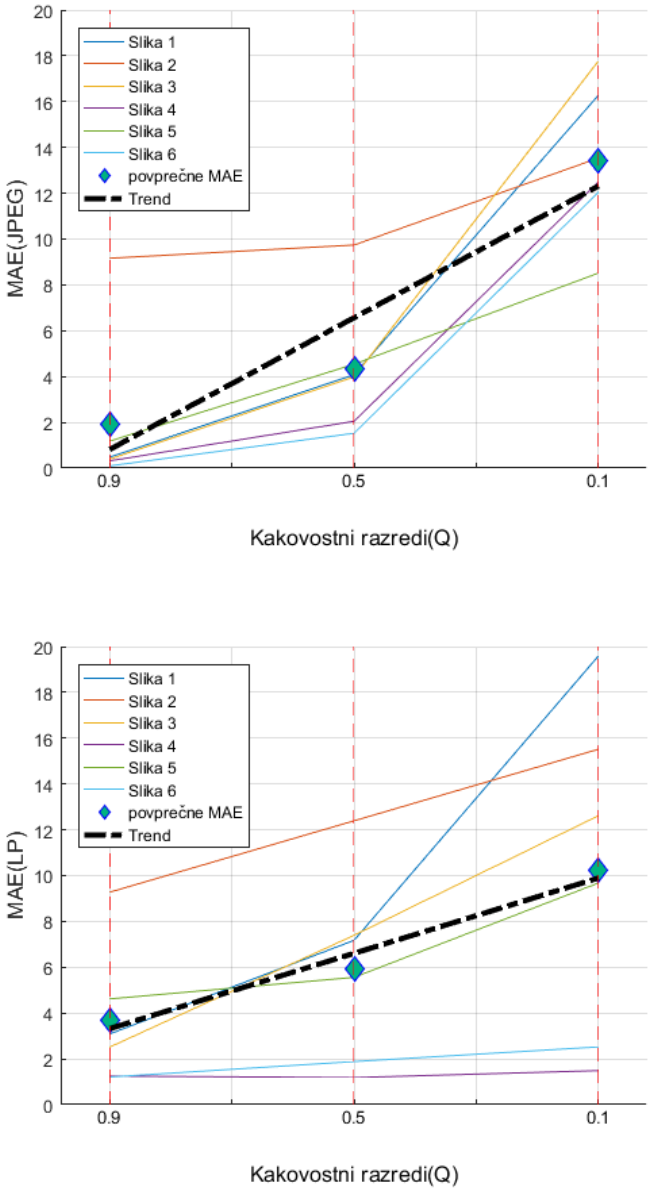
(j) Slika 4

(k) Slika 5 (JPEG)

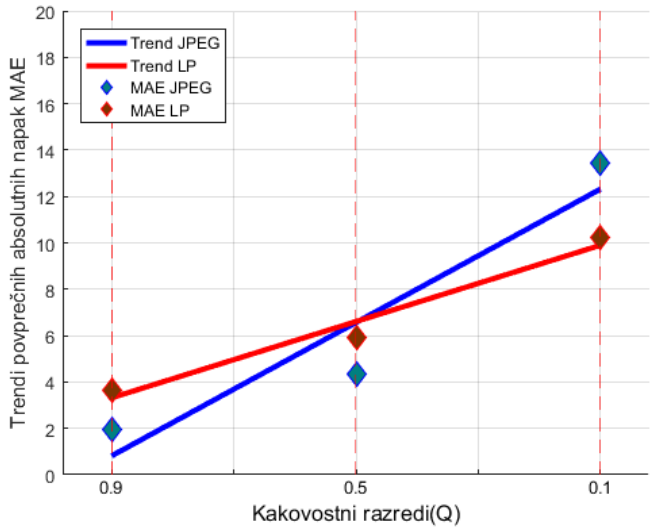
Slika 4.4: Nizka kakovost

(a) JPEG (b) LP

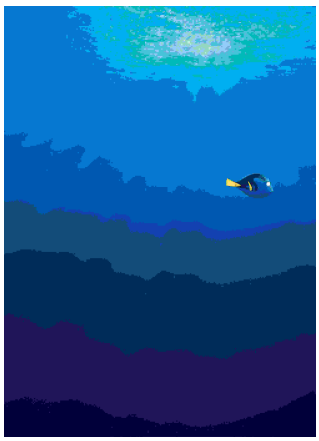
Slika 4.5: Primerjava Slike 4 stisnjene z obema algoritmoma pri nizki kakovosti



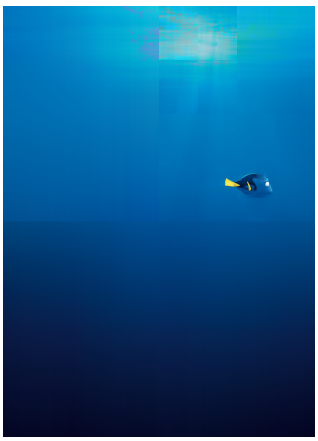
Slika 4.6: Trendi stiskanja slik s pomočjo metode JPEG(zgoraj) in s pomočjo metode LP(spodaj).



Slika 4.7: Trendi stiskanja slik z metodo JPEG in metodo LP.



(a) JPEG



(b) LP

Slika 4.8: Primerjava Slike 6 stisnjene z obema algoritmoma pri nizki kakovosti

Poglavje 5

Zaključek

Predstavili smo algoritem za kompresijo slik s pomočjo linearne regresije in delitve slik na ploske sektorje, ter njen inverzni komplementarni algoritem za dekompresijo. S temi orodji lahko kompresiramo poljubne slike poljubnih velikosti, a izkaže se, da je algoritem najbolj učinkovit, ko pride do risanih slik, kot so posamezne sličice v risankah. Implementacija algoritma v programskem jeziku *Java* je dostopna na avtorjevem spletnem repozitoriju [13].

5.0.1 Izboljšave in nadaljnje delo

Na predstavljenem algoritmu aktivno delamo, ga popravljamo in izboljšujemo. Delo je hkrati hobi in želja po boljšem kompresijskem algoritmu. Delo je zelo zanimivo, razvijanje in implementiranje znanih metod v drugačnem problemskem prostoru pa je zasvojljive narave, zato bomo idejo v prihodnosti še razvijali. V nadaljevanju so opisane načrtovane izboljšave.

Prileżne funkcije višje stopnje

Algoritem je omejen na prileganje premic na podatke. A prileżna funkcija, ki jo iščemo, bi lahko bila polinom višje stopnje, s katero bi osnovne podatke bolje opisali. Zato pa potrebujemo cenilke, ki bodo višje momente dobro opisale.

Primernejši model YCbCr

Zaenkrat smo za barvni model uporabili konstantno transformacijo. Lahko pa bi primerno bazo za barvni model slike našli s pomočjo analize glavnih komponent PCA [18]. PCA je metoda iskanja ortogonalnih baznih vektorjev prostora, ki maksimizirajo varianco, hkrati pa kovarianco minimizirajo. Za barvni model bi zato lahko poiskali takšne ortogonalne bazne vektorje, ki bi maksimizirali podrobnosti na enem samem kanalu, medtem ko bi na drugih dveh podrobnosti zmanjšali, saj bi lahko s tem kanale tudi bolj stisnili.

Kvaliteta signalov

V splošnem velja, da je naše oko bolj občutljivo na spremembe svetlosti, kot na spremembe barvitosti [1]. Zato bi lahko podvzorčili signale barvitosti v različnih shemah učinkovitosti, saj bi nam to pomenilo drastične razlike v velikosti kodirane slike, hkrati pa razlike v kakovosti ne bi bile očitne.

Naravna sektorizacija

Sektorizacija, opisana v diplomskem delu, poteka po naivno izračunanih mejah. Slike imajo sektorje ploskih kvadrantov razporejene po vzorcu vsebine slike. Z naravnim sektoriziranjem, kjer bi sliko razrezali tam, kjer bi to pomenilo najvišjo učinkovitost, bi lahko velikost kompresirane slike še zmanjšali.

Literatura

[1] Chrominance subsampling in digital images. Dosegljivo: `http://dougkerr.net/Pumpkin/articles/Subsampling.pdf`. [Dostopano: 29.10.2017].

[2] Smith Vivianne C and Pokorny Joel. Spectral sensitivity of the foveal cone photopigments between 400 and 500 nm. *Vision research*, 15(2):161–171, 1975.

[3] Centpacrr. Lenna. Dosegljivo: `https://upload.wikimedia.org/wikipedia/en/2/24/Lenna.png`, 2005. [Dostopano: 29.10.2017].

[4] Allen Gersho and Robert M Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.

[5] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[6] Robert M Gray et al. Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239, 2006.

[7] Gerald C Holst and Terrence S Lomheim. *CMOS/CCD sensors and camera systems*, volume 408. JCD publishing USA, 2007.

[8] Žiga Sajovic. Operatorski račun nad programskimi prostori. 2016. Diplomsko delo.

[9] Žiga Sajovic and et al. Operational calculus on programming spaces. arXiv:1610.07690, 2016.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[11] John Miano. *Compressed image file formats: Jpeg, png, gif, xbm, bmp*. Addison-Wesley Professional, 1999.

[12] Christoph Peters. Ycbcrcolorspace perspective. Dosegljivo: `https://upload.wikimedia.org/wikipedia/en/f/f4/YCbCrColorSpace_Perspective.png`, 2011. [Dostopano: 29.10.2017].

[13] Gašper Primožič. Image compression with linear regression and sectorisation, 2016. https://github.com/gapox/ImageCompression.

[14] RaphaelQS. Linear regression. Dosegljivo: `https://upload.wikimedia.org/wikipedia/commons/3/3a/Linear_regression.svg`, 2016. [Dostopano: 29.10.2017].

[15] Howard J. Seltman. Experimental design and analysis. Dosegljivo: `http://www.stat.cmu.edu/~hseltman/309/Book/Book.pdf`, 2015. [Dostopano: 15.09.2017].

[16] Welch Terry. A technique for high-performance data compression. Dosegljivo: `http://www.cs.duke.edu/courses/spring03/cps296.5/papers/welch_1984_technique_for.pdf`, 1984.

[17] Nancy Martha West. *Kodak and the Lens of Nostalgia*. University of virginia Press, 2000.

[18] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.